TASK_4_SALES_PREDICTION_USING_PYTHON

1. Loading the dataset

[1]:

```python
import pandas as pd

# GitHub raw URL
url = 'https://raw.githubusercontent.com/abuthahir17/CODSOFT_INTERNSHIP/main/
 ↪advertising.csv'

# Read CSV file
data = pd.read_csv(url)

print("Dataset loaded successfully.")
```

```
Dataset loaded successfully.
```

2. Data Inspection

2.1. First 5 Rows

[2]:

```python
print("First 5 rows of the dataset:")
print(data.head())
```

```
First 5 rows of the dataset:
      TV  Radio  Newspaper  Sales
0  230.1   37.8       69.2   22.1
1   44.5   39.3       45.1   10.4
2   17.2   45.9       69.3   12.0
3  151.5   41.3       58.5   16.5
4  180.8   10.8       58.4   17.9
```

2.2. Dataset Information

[3]:

```python
print("Information about the dataset:")
print(data.info())
```

```
Information about the dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
```

```
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
None
```

2.3. Describing the Dataset

[4]:

```
print("Descriptive statistics of the dataset:")
print(data.describe())
```

```
Descriptive statistics of the dataset:
               TV       Radio   Newspaper       Sales
count  200.000000  200.000000  200.000000  200.000000
mean   147.042500   23.264000   30.554000   15.130500
std     85.854236   14.846809   21.778621    5.283892
min      0.700000    0.000000    0.300000    1.600000
25%     74.375000    9.975000   12.750000   11.000000
50%    149.750000   22.900000   25.750000   16.000000
75%    218.825000   36.525000   45.100000   19.050000
max    296.400000   49.600000  114.000000   27.000000
```

2.4. Checking Dataset Shape

[5]:

```
print("Dataset Shape:", data.shape)
```

```
Dataset Shape: (200, 4)
```

2.5. Checking Missing Values

[6]:

```
print("Missing values in each column:")
print(data.isnull().sum())

# There is no missing value
```

```
Missing values in each column:
TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64
```

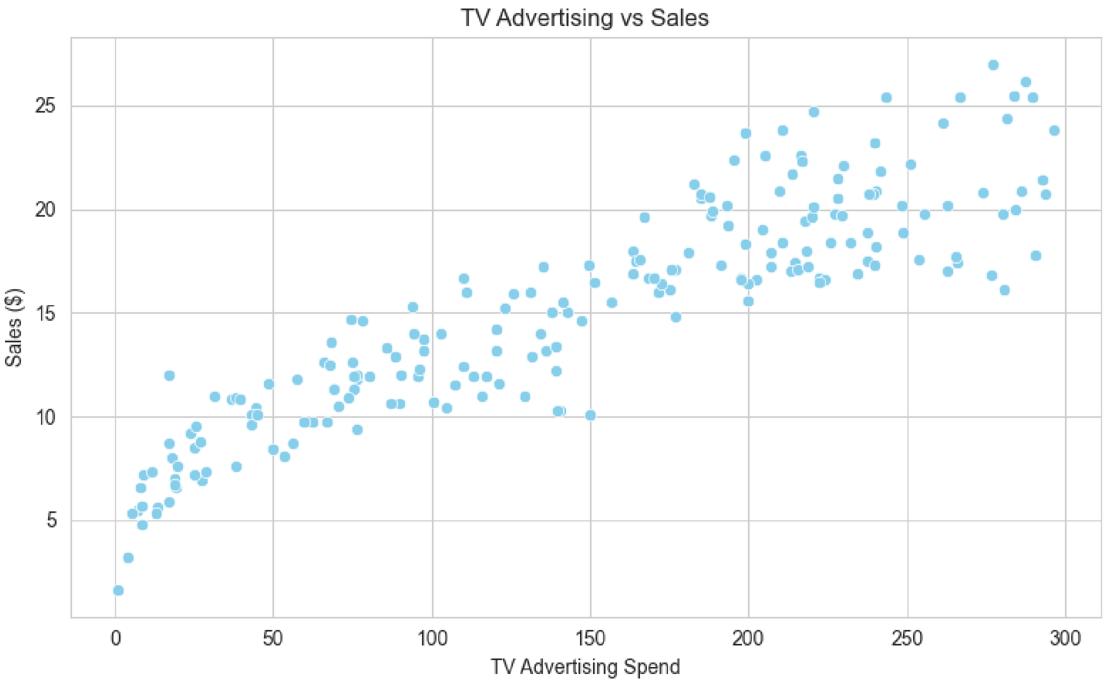
3. Data Visualization

3.1. TV Advertising vs. Sales Scatter Plot

[7]:

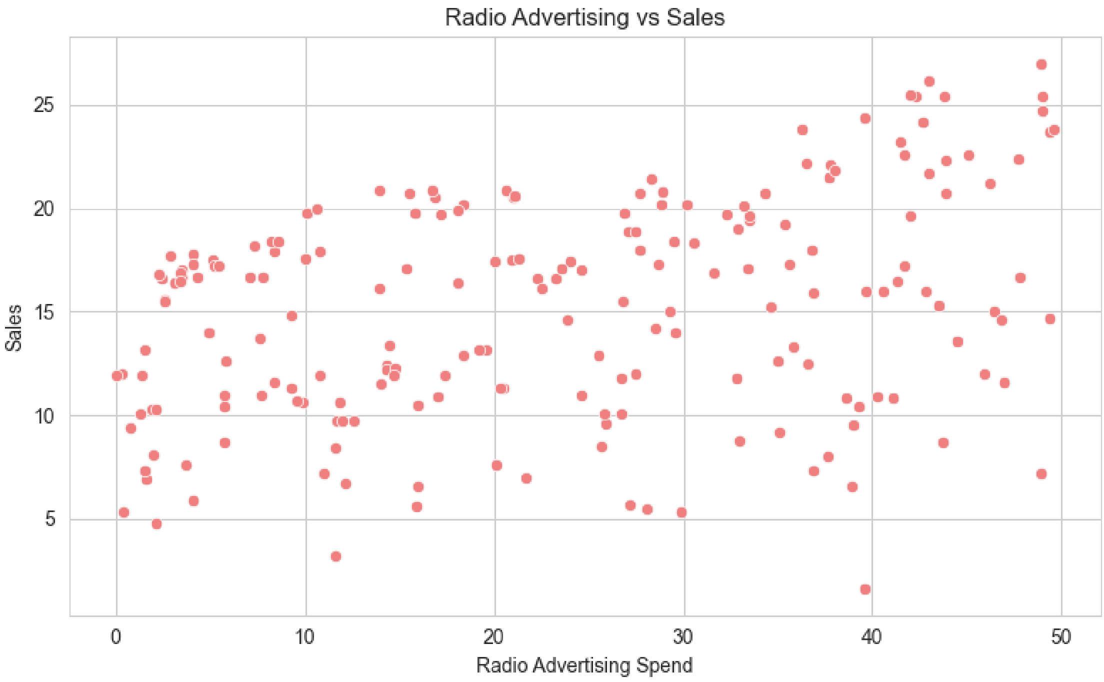
```
import seaborn as sns
import matplotlib.pyplot as plt

# Set seaborn style
```

```
sns.set_style("whitegrid")

# TV vs Sales
plt.figure(figsize=(8, 5))
sns.scatterplot(x='TV', y='Sales', data=data, color='skyblue')
plt.title('TV Advertising vs Sales')
plt.xlabel('TV Advertising Spend')
plt.ylabel('Sales ($)')
plt.grid(True)
plt.tight_layout()
plt.show()
```



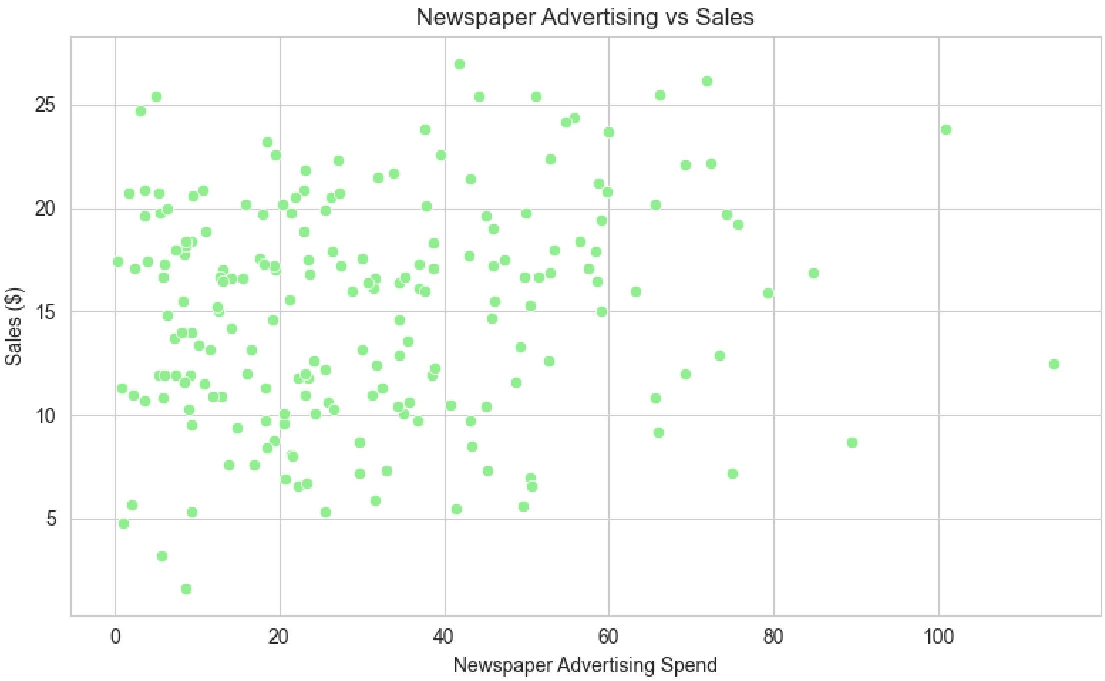
3.2. Radio Advertising vs. Sales Scatter Plot

[8]:
```
# Radio vs Sales
plt.figure(figsize=(8,5))
sns.scatterplot(x='Radio', y='Sales', data=data, color='lightcoral')
plt.title('Radio Advertising vs Sales')
plt.xlabel('Radio Advertising Spend')
plt.ylabel('Sales')
plt.grid(True)
plt.tight_layout()
plt.show()
```



3.3. Newspaper Advertising vs. Sales Scatter Plot

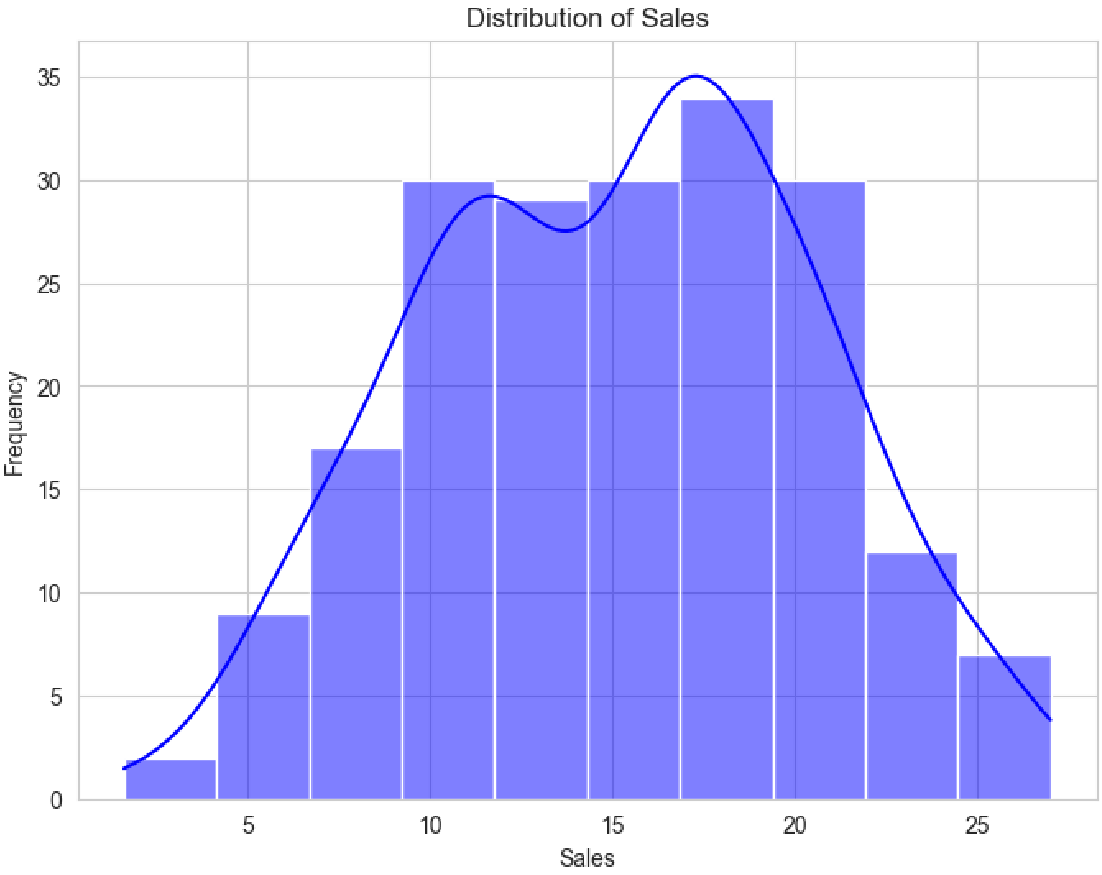
[9]:
```
# Newspaper vs Sales
plt.figure(figsize=(8, 5))
sns.scatterplot(x='Newspaper', y='Sales', data=data, color='lightgreen')
plt.title('Newspaper Advertising vs Sales')
plt.xlabel('Newspaper Advertising Spend')
plt.ylabel('Sales ($)')
plt.grid(True)
plt.tight_layout()
plt.show()
```



3.4. Distribution of Sales Histogram

[10]:

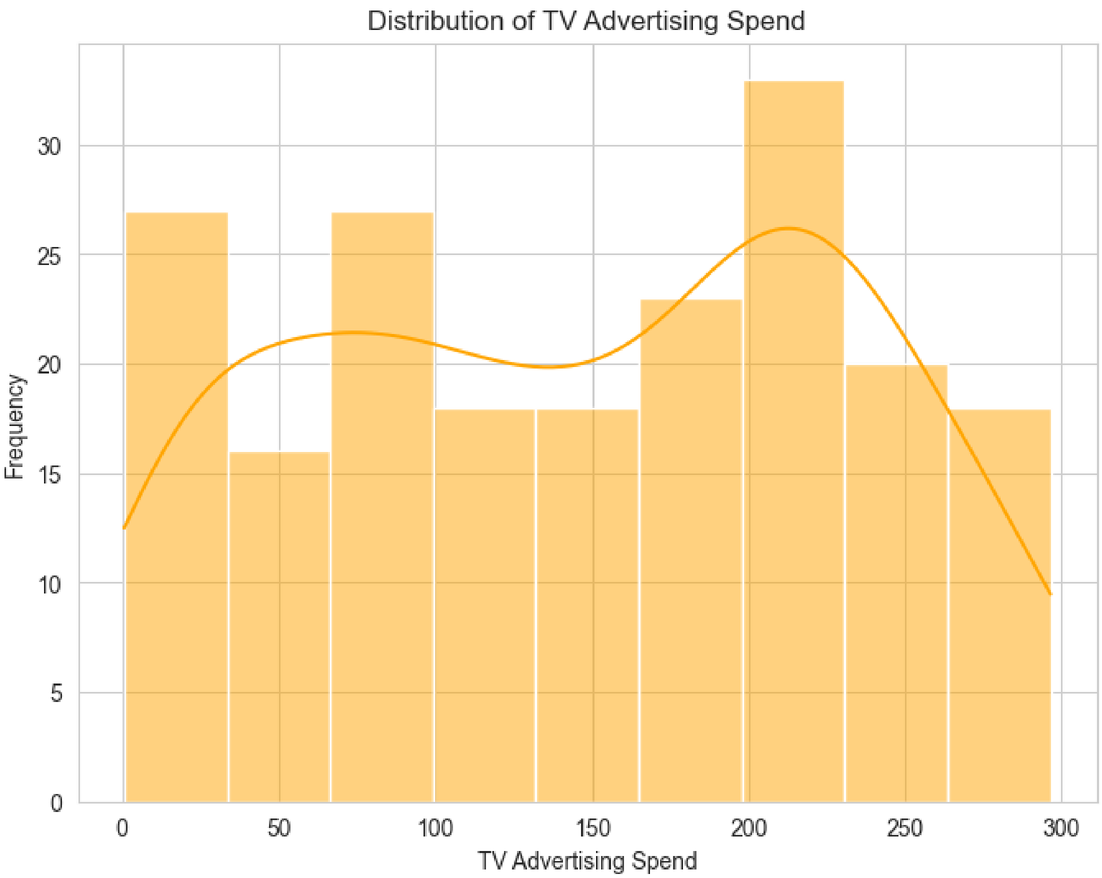
```
# Distribution of Sales
plt.figure(figsize=(8, 6))
sns.histplot(data['Sales'], kde=True, color='blue')
plt.title('Distribution of Sales')
plt.xlabel('Sales')
plt.ylabel('Frequency')
plt.show()
```



3.5. Distribution of TV Advertising Spend Histogram

[11]:

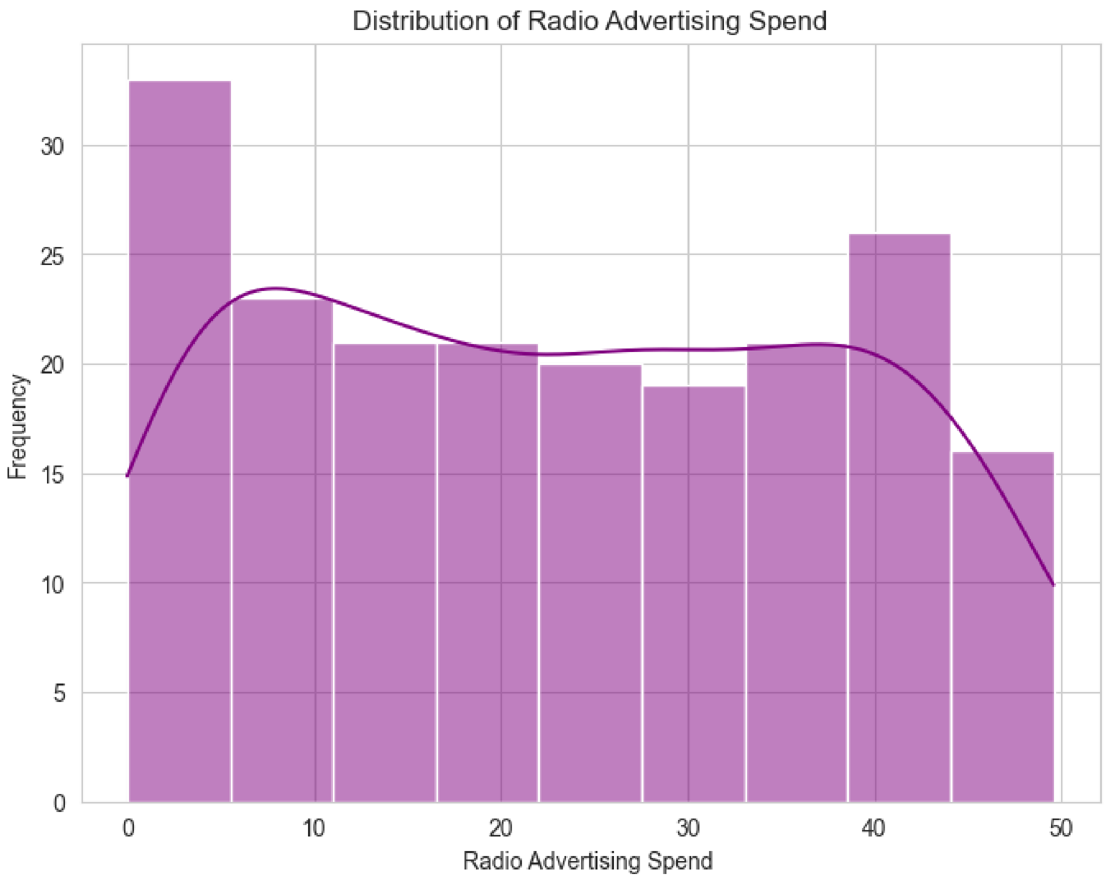
```
# Distribution of TV
plt.figure(figsize=(8, 6))
sns.histplot(data['TV'], kde=True, color = "orange")
plt.title("Distribution of TV Advertising Spend")
plt.xlabel("TV Advertising Spend")
plt.ylabel("Frequency")
plt.show()
```



3.6. Distribution of Radio Advertising Spend Histogram

[12]:

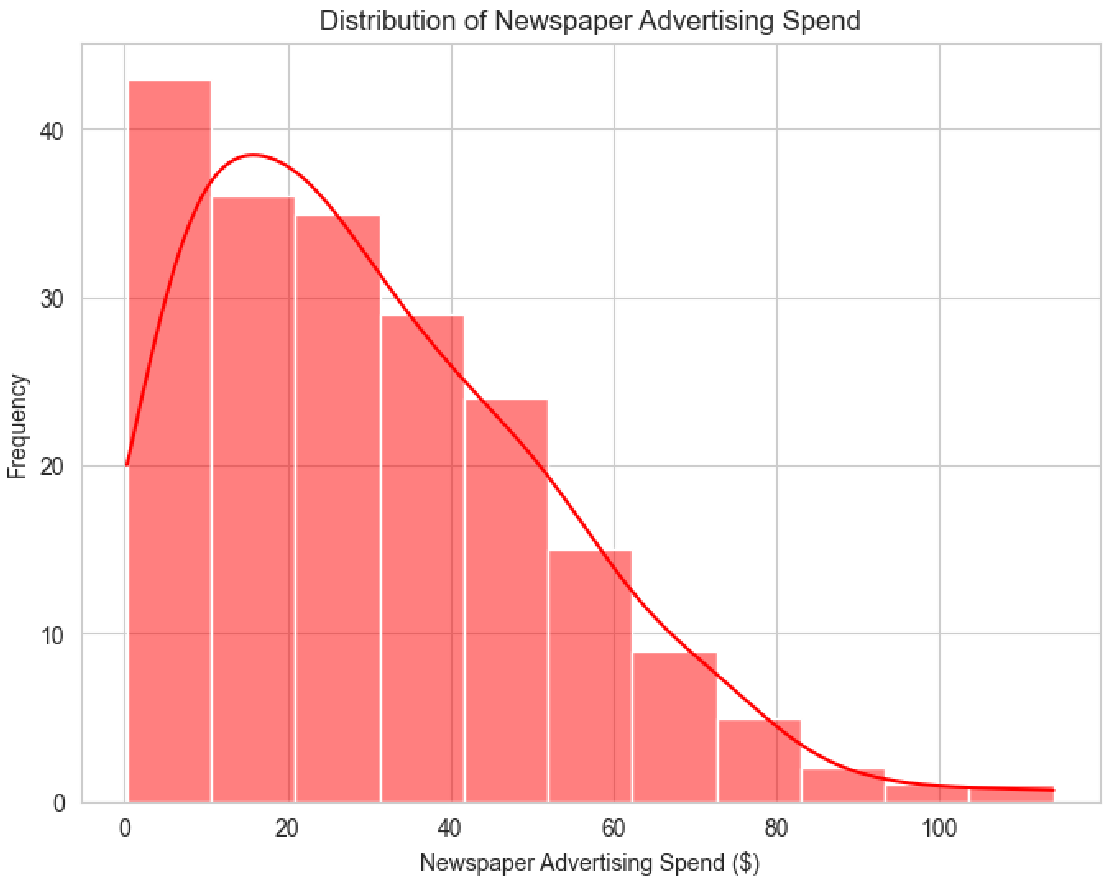
```
# Distribution of Radio
plt.figure(figsize=(8, 6))
sns.histplot(data['Radio'], kde=True, color = "purple")
plt.title("Distribution of Radio Advertising Spend")
plt.xlabel("Radio Advertising Spend ")
plt.ylabel("Frequency")
plt.show()
```



3.7. Distribution of Newspaper Advertising Spend Histogram

[13]:

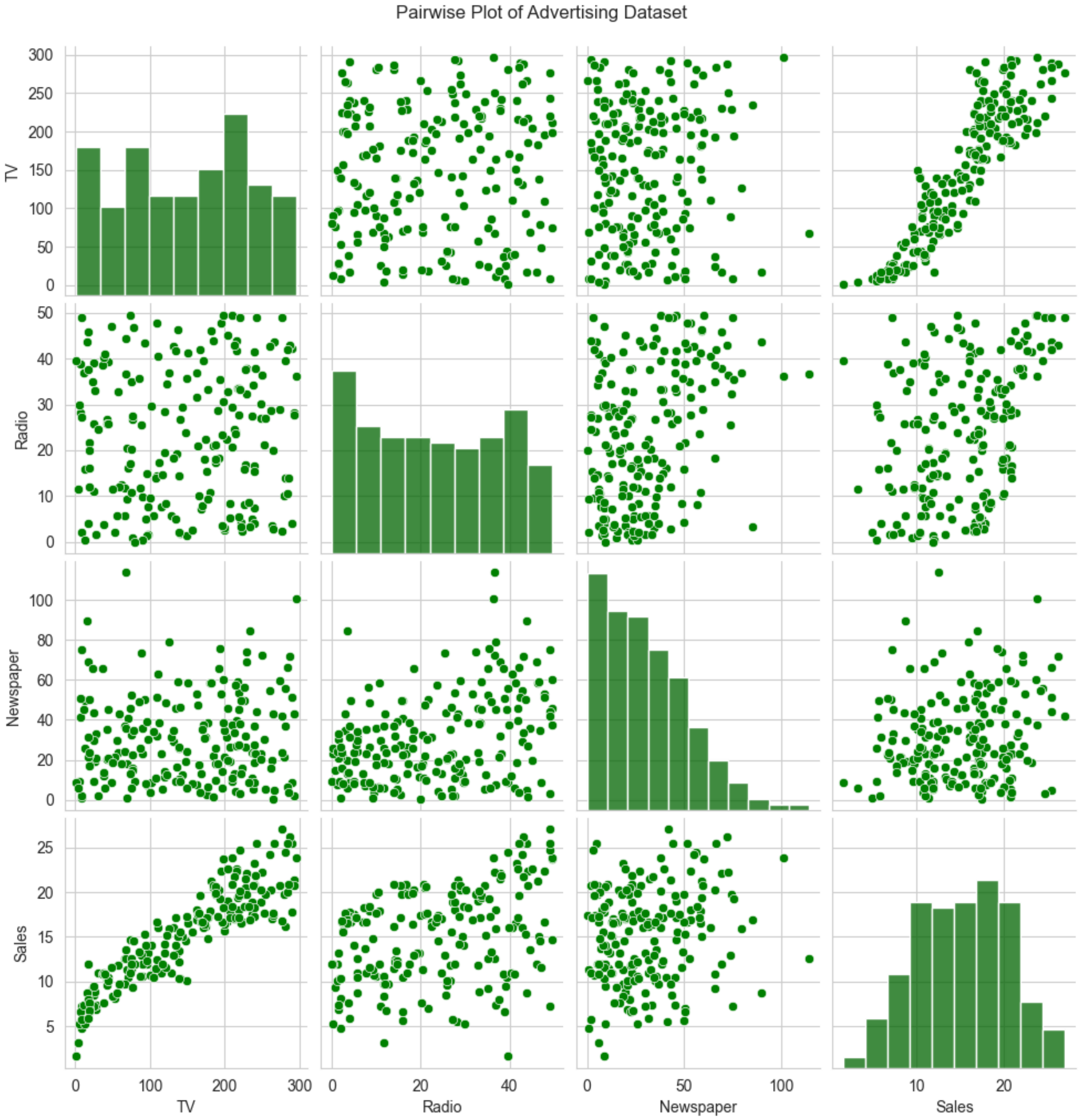
```
# Distribution of Newspaper
plt.figure(figsize=(8, 6))
sns.histplot(data['Newspaper'], kde=True, color = "red")
plt.title("Distribution of Newspaper Advertising Spend")
plt.xlabel("Newspaper Advertising Spend ($)")
plt.ylabel("Frequency")
plt.show()
```



3.8. Pairwise Plot of Advertising Dataset

[14]:

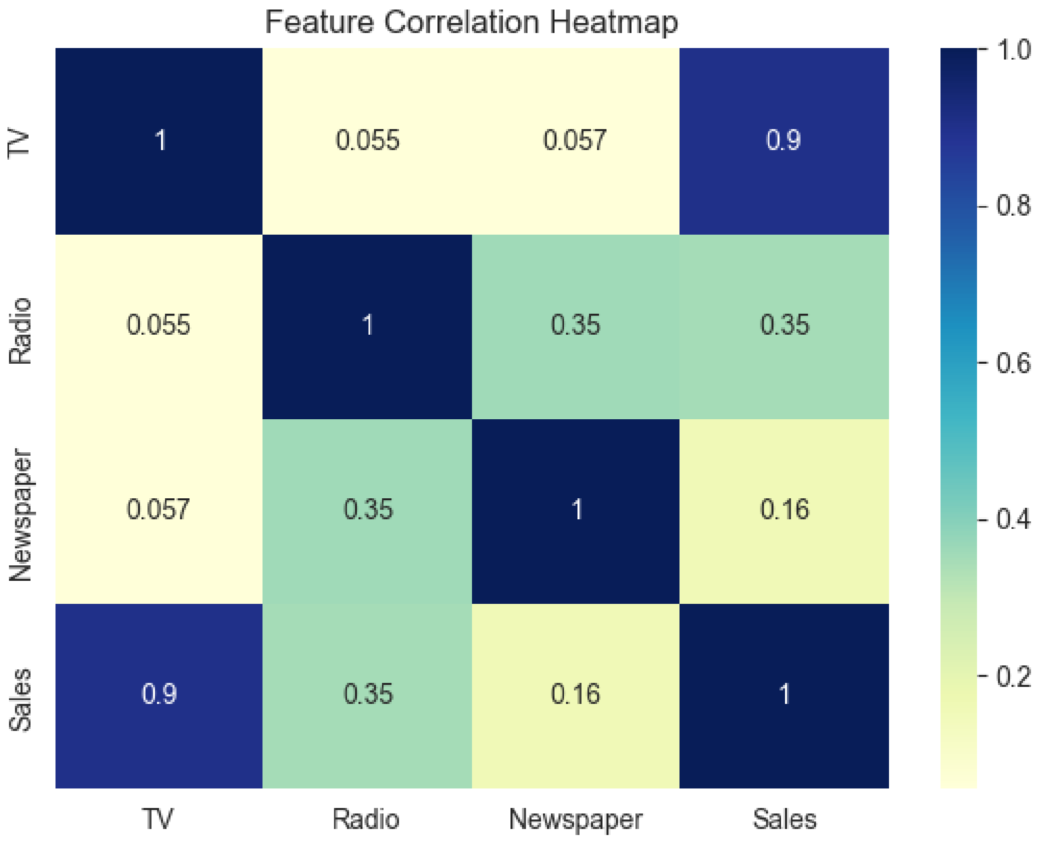
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.pairplot(data,diag_kws={'color': 'darkgreen', 'fill': True},␣
 ↪plot_kws={'color': 'green'})
plt.suptitle("Pairwise Plot of Advertising Dataset", y=1.02)
plt.show()
```



3.9. Feature Correlation Heatmap

```
[15]: print("Correlation Heatmap:\n")
      plt.figure(figsize=(7, 5))
      sns.heatmap(data.corr(), annot=True, cmap='YlGnBu')
      plt.title("Feature Correlation Heatmap")
      plt.show()
```

```
Correlation Heatmap:
```



4. Data Preprocessing

4.1. Feature and Target Separation

[16]:

```
# Define features (X) and target (y)
# Split the data
X = data[['TV', 'Radio', 'Newspaper']]
y = data['Sales'] # Target variable
```

4.2. Data Splitting

[17]:

```
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

print(f"\nTraining set size: {X_train.shape[0]} samples")
print(f"Testing set size: {X_test.shape[0]} samples")
```

```
Training set size: 160 samples
Testing set size: 40 samples
```

5. Model Training

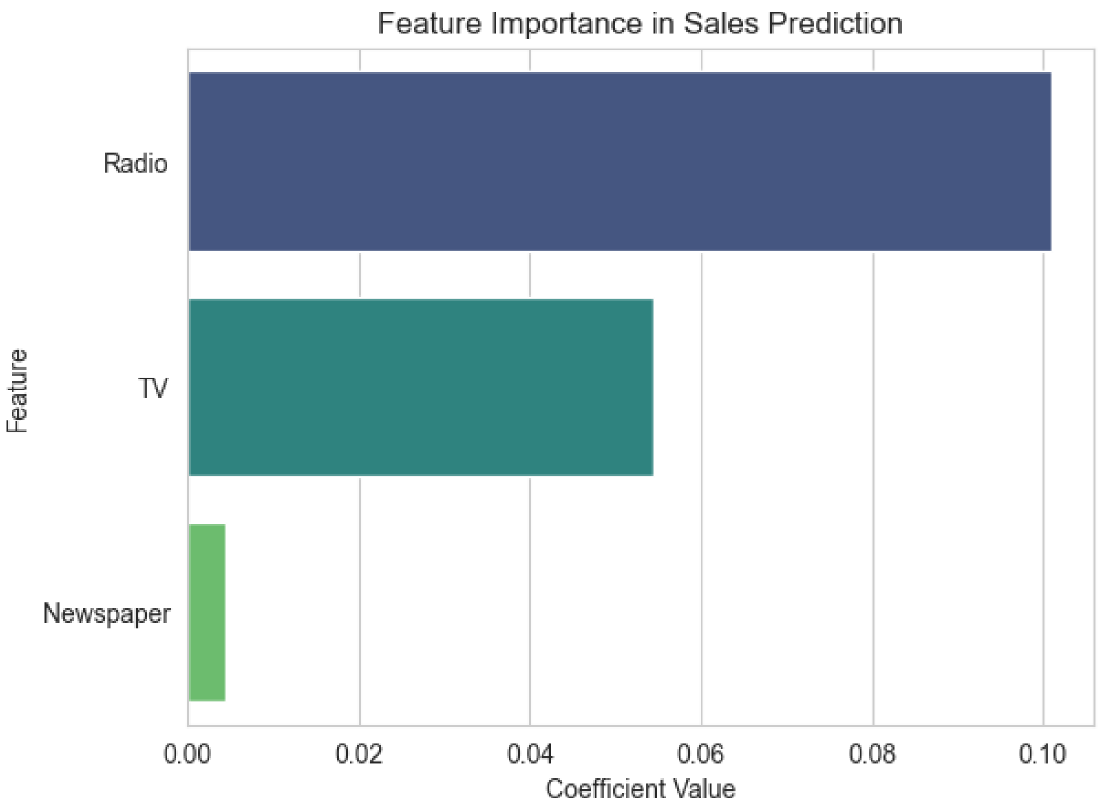
5.1. Initialize the models

[18]:
```
from sklearn.linear_model import LinearRegression
# Initialize the Linear Regression model
# This uses a 'machine learning technique' (Linear Regression) to predict sales.
model = LinearRegression()

# Train the model using the training data
model.fit(X_train, y_train)

print("Model training complete.")
```

```
Model training complete.
```

5.2. Feature Importance

[19]:
```
# Coefficients of the trained Linear Regression model
coefficients = pd.DataFrame({ 'Feature': X.columns, 'Coefficient': model.coef_␣
 ↪}).sort_values(by='Coefficient', ascending=False)

# Bar chart of coefficients
sns.barplot(x='Coefficient', y='Feature', data=coefficients, palette='viridis',␣
 ↪ hue='Feature')
plt.title("Feature Importance in Sales Prediction")
plt.xlabel("Coefficient Value")
plt.ylabel("Feature")
plt.show()
```



5.3. Model Prediction

[20]:

```
y_pred = model.predict(X_test)
```

6. Model Evaluation

6.1. Calculating Regression Model Performance

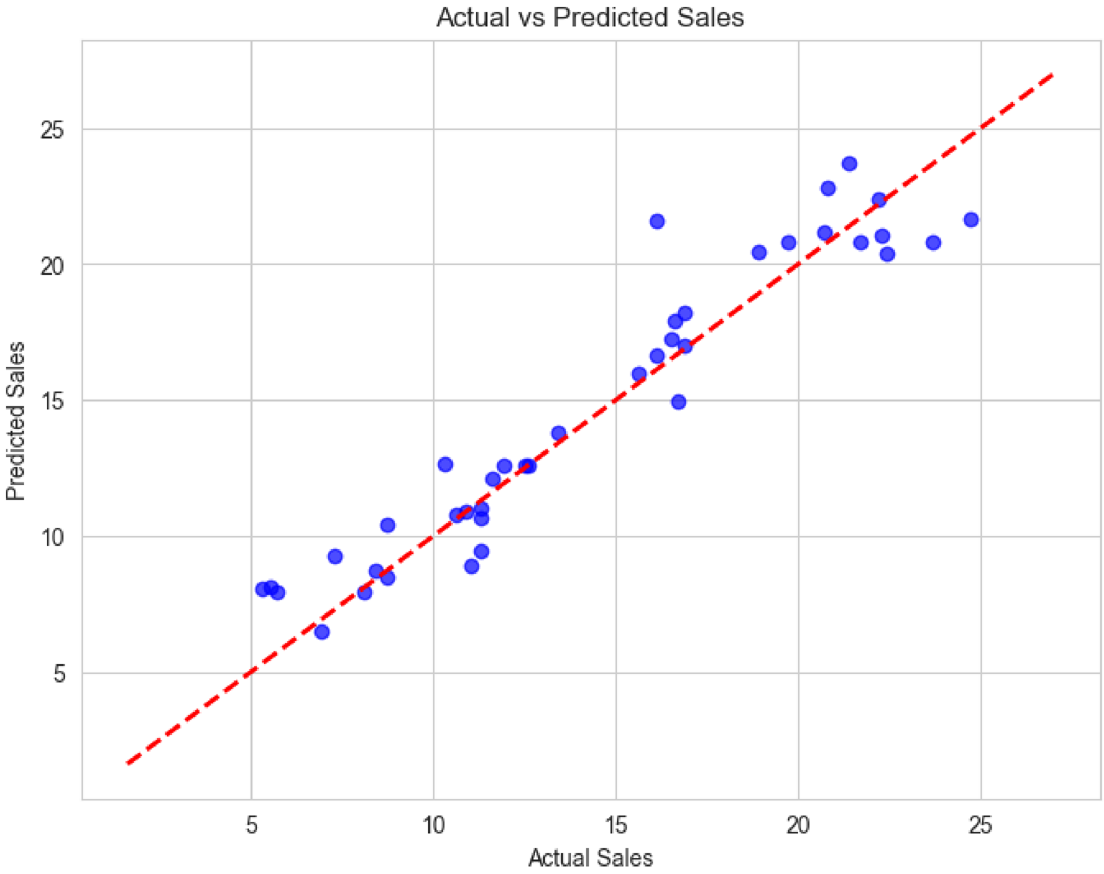
[21]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
# Calculate evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R2): {r2:.2f}")

print(f"\nModel Accuracy (based on R-squared): {r2 * 100:.2f}%")
```

```
Mean Absolute Error (MAE): 1.27
Mean Squared Error (MSE): 2.91
Root Mean Squared Error (RMSE): 1.71
R-squared (R2): 0.91

Model Accuracy (based on R-squared): 90.59%
```

6.2. Visualizing Actual vs. Predicted Sales

[22]:
```
# Plotting actual vs predicted values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color='blue')
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2) # Diagonal line
plt.xlabel('Actual Sales')
plt.ylabel('Predicted Sales')
plt.title('Actual vs Predicted Sales')
plt.show()
```



7. Boosting Model Performance (Accuracy)

7.1. Creating a Binary Sales Target to improve Accuracy

[23]:

```
# Convert 'Sales' into a binary classification target: High Sales (1) or Low␣
 ↪Sales (0)
data['Sales_Class'] = (data['Sales'] > data['Sales'].median()).astype(int)
```

7.2. Feature and Target Variable Definition

[24]:

```
# Features and classification target
X = data[['TV', 'Radio', 'Newspaper']]
y = data['Sales_Class']
```

7.3. Splitting Data into Training and Testing Sets

[25]:

```
# Split into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

7.4. Random Forest Classifier: Training and Evaluation

[26]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
# Random Forest Classifier
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
acc_rf = accuracy_score(y_test, y_pred_rf)
cm_rf = confusion_matrix(y_test, y_pred_rf)
```

7.5. Logistic Regression: Training and Evaluation

[27]:

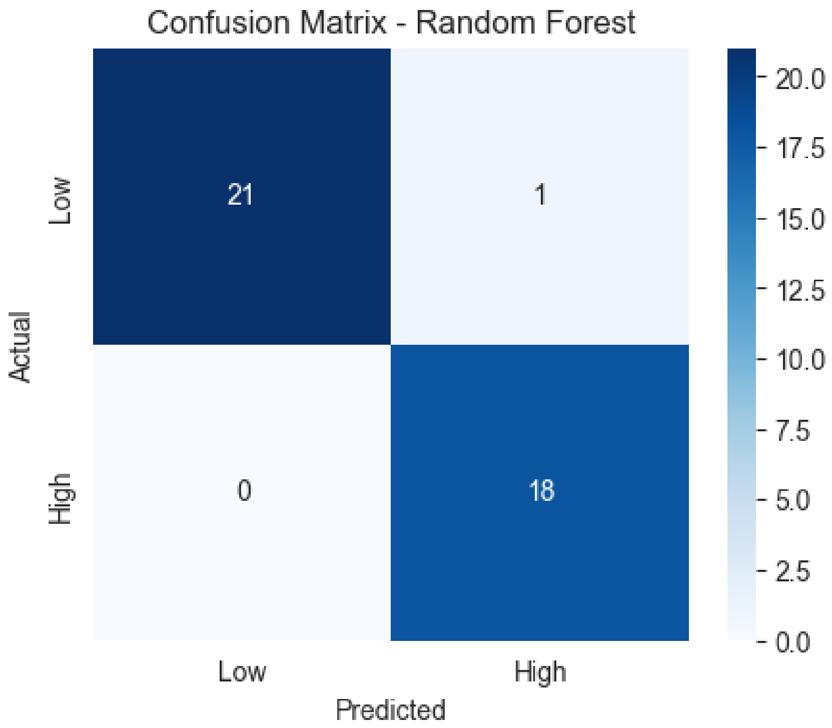
```
# Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred_log = logreg.predict(X_test)
acc_log = accuracy_score(y_test, y_pred_log)
cm_log = confusion_matrix(y_test, y_pred_log)
```

7.6. Displaying Random Forest Performance and Confusion Matrix

[28]:

```
print("Random Forest Classifier:")
print(f"Accuracy: {acc_rf * 100 :.2f}%")

print("\nConfusion Matrix for Random Forest:\n")
plt.figure(figsize=(5,4))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', xticklabels=['Low',␣
 ↪'High'], yticklabels=['Low', 'High'])
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

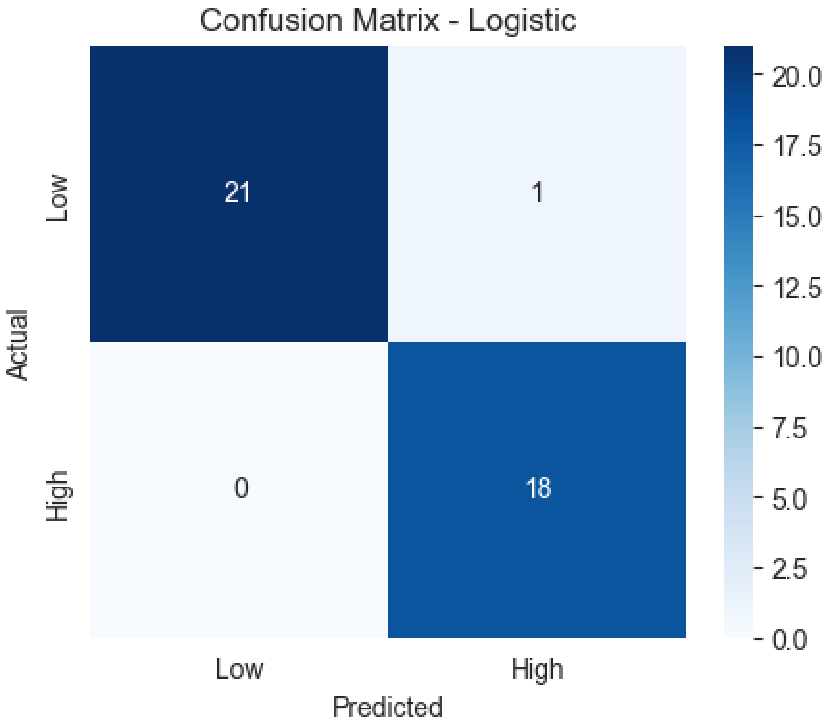
```
Random Forest Classifier:
Accuracy: 97.50%

Confusion Matrix for Random Forest:
```



7.7. Displaying Logistic Regression Performance and Confusion Matrix

```
[29]: print("Logistic Regression:")
print(f"Accuracy: {acc_log * 100 :.2f}%")
print("\nConfusion Matrix for Logistic Regression:\n")
plt.figure(figsize=(5,4))
sns.heatmap(cm_log, annot=True, fmt='d', cmap='Blues', xticklabels=['Low',␣
 ↪'High'], yticklabels=['Low', 'High'])
plt.title('Confusion Matrix - Logistic')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Logistic Regression:
Accuracy: 97.50%

Confusion Matrix for Logistic Regression:
```



8. Sales Prediction Example

8.1. Example 1: Forecasting Sales with Specific Advertising Spends

[30]:

```
# Example: Predict sales for new advertising expenditures
# Predict sales based on a specific advertising budget

new_ad_spend = pd.DataFrame([[200, 30, 40]], columns=['TV', 'Radio',␣
  ↪'Newspaper'])
predicted_sales = model.predict(new_ad_spend)

print(f"Advertising Spend (TV: 200, Radio: 30, Newspaper: 40):")
print(f"Predicted Sales: {predicted_sales[0]:.2f}")
```

```
Advertising Spend (TV: 200, Radio: 30, Newspaper: 40):
Predicted Sales: 18.82
```

8.2. Example 2: Sales Prediction for an Alternative Advertising Scenario

[31]:

```
# Another example
# Predict sales based on a specific advertising budget

new_ad_spend_2 = pd.DataFrame([[50, 10, 5]], columns=['TV', 'Radio',␣
  ↪'Newspaper'])
predicted_sales_2 = model.predict(new_ad_spend_2)
```

```
print(f"\nAdvertising Spend (TV: 50, Radio: 10, Newspaper: 5):")
print(f"Predicted Sales: {predicted_sales_2[0]:.2f}")
```

```
Advertising Spend (TV: 50, Radio: 10, Newspaper: 5):
Predicted Sales: 8.47
```

[32]:

```
print("--- Sales Prediction Project Complete ---")
```

```
--- Sales Prediction Project Complete ---
```